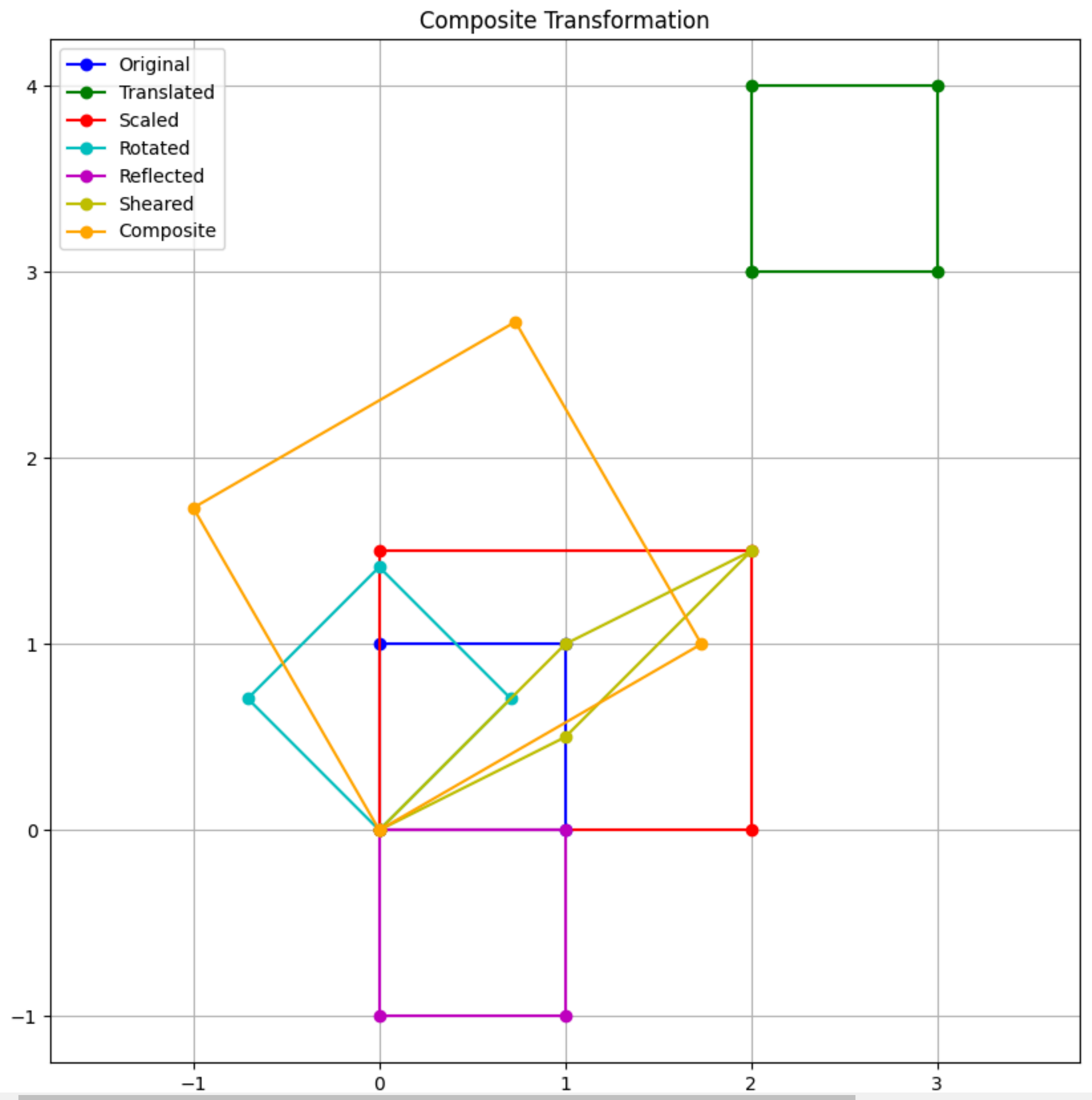
```
import numpy as np
import matplotlib.pyplot as plt

def plot_polygon(vertices, title="Polygon", color='b'):
    """Plot the given polygon"""
    vertices = np.append(vertices, [vertices[0]], axis=0)  # To close the polygon
    plt.plot(vertices[:, 0], vertices[:, 1], marker='o', color=color)
    plt.title(title)
    plt.axis('equal')
    plt.grid(True)


def translate(vertices, tx, ty):
    """Translate the polygon by tx and ty"""
    translation_matrix = np.array([[1, 0, tx],
                                   [0, 1, ty],
                                   [0, 0, 1]])
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])  # Homogeneous coordinates
    transformed_vertices = vertices_h @ translation_matrix.T
    return transformed_vertices[:, :2]


def scale(vertices, sx, sy):
    """Scale the polygon by sx and sy"""
    scaling_matrix = np.array([[sx, 0, 0],
                               [0, sy, 0],
                               [0,  0, 1]])
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])
    transformed_vertices = vertices_h @ scaling_matrix.T
    return transformed_vertices[:, :2]


def rotate(vertices, angle):
    """Rotate the polygon by the given angle (in degrees)"""
    radians = np.radians(angle)
    rotation_matrix = np.array([[np.cos(radians), -np.sin(radians), 0],
                                [np.sin(radians),  np.cos(radians), 0],
                                [0,                0,               1]])
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])
    transformed_vertices = vertices_h @ rotation_matrix.T
    return transformed_vertices[:, :2]


def reflect(vertices, axis):
    """Reflect the polygon across the specified axis ('x' or 'y')"""
    if axis == 'x':
        reflection_matrix = np.array([[1,  0, 0],
                                      [0, -1, 0],
                                      [0,  0, 1]])
    elif axis == 'y':
        reflection_matrix = np.array([[-1, 0, 0],
                                      [0,  1, 0],
                                      [0,  0, 1]])
    else:
        raise ValueError("Axis must be 'x' or 'y'")
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])
    transformed_vertices = vertices_h @ reflection_matrix.T
    return transformed_vertices[:, :2]


def shear(vertices, shx=0, shy=0):
    """Shear the polygon by shx (x-axis shear) and shy (y-axis shear)"""
    shearing_matrix = np.array([[1, shx, 0],
                                [shy, 1, 0],
                                [0,  0, 1]])
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])
    transformed_vertices = vertices_h @ shearing_matrix.T
    return transformed_vertices[:, :2]


def composite_transform(vertices, transformations):
    """Apply a composite transformation to the polygon"""
    composite_matrix = np.eye(3)  # Identity matrix
    for transform in transformations:
        composite_matrix = composite_matrix @ transform  # Combine transformations
    vertices_h = np.hstack([vertices, np.ones((vertices.shape[0], 1))])
    transformed_vertices = vertices_h @ composite_matrix.T
    return transformed_vertices[:, :2]

# Example usage
if __name__ == "__main__":
```

```
# Define a square polygon
square = np.array([[0, 0], [1, 0], [1, 1], [0, 1]])

plt.figure(figsize=(10, 10))

# Original polygon
plot_polygon(square, "Original Polygon", color='b')

# Translation
translated_square = translate(square, 2, 3)
plot_polygon(translated_square, "Translated Polygon", color='g')

# Scaling
scaled_square = scale(square, 2, 1.5)
plot_polygon(scaled_square, "Scaled Polygon", color='r')

# Rotation
rotated_square = rotate(square, 45)
plot_polygon(rotated_square, "Rotated Polygon", color='c')

# Reflection
reflected_square = reflect(square, 'x')
plot_polygon(reflected_square, "Reflected Polygon (X-Axis)", color='m')

# Shearing
sheared_square = shear(square, shx=1, shy=0.5)
plot_polygon(sheared_square, "Sheared Polygon", color='y')

# Composite transformation (scale and rotate)
scale_matrix = np.array([[2, 0, 0],
                          [0, 2, 0],
                          [0, 0, 1]])
rotation_matrix = np.array([[np.cos(np.radians(30)), -np.sin(np.radians(30)), 0],
                             [np.sin(np.radians(30)),  np.cos(np.radians(30)), 0],
                             [0,                       0,                       1]])
composite_square = composite_transform(square, [scale_matrix, rotation_matrix])
plot_polygon(composite_square, "Composite Transformation", color='orange')

plt.legend(["Original", "Translated", "Scaled", "Rotated", "Reflected", "Sheared", "Composite"],
           loc='upper left')
plt.show()
```



```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Function to display an image in Colab
def show_image(title, image):
    plt.figure(figsize=(8, 6))
    plt.title(title)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct color display
    plt.axis('off')
    plt.show()

# Load an image (Make sure to upload an image or use one from a URL)
from google.colab import files
uploaded = files.upload()  # Upload your image
image_path = list(uploaded.keys())[0]
image = cv2.imread(image_path)
show_image("Original Image", image)

# Scaling Function
def scale_image(image, fx, fy):
    if fx <= 0 or fy <= 0:
        raise ValueError("Scaling factors (fx and fy) must be positive.")
    return cv2.resize(image, None, fx=fx, fy=fy, interpolation=cv2.INTER_LINEAR)

# Example Scaling
scaled_up = scale_image(image, 1.5, 1.5)
show_image("Scaled Image (1.5x)", scaled_up)

scaled_down = scale_image(image, 0.5, 0.5)
show_image("Scaled Image (0.5x)", scaled_down)

# Translation Function
def translate_image(image, tx, ty):
    rows, cols = image.shape[:2]
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    return cv2.warpAffine(image, M, (cols, rows))
```

```
        return cv2.warpAffine(image, M, (cols, rows))

    translated_image = translate_image(image, 100, 50)
    show_image("Translated Image (tx=100, ty=50)", translated_image)

    # Rotation Function
    def rotate_image(image, angle):
        rows, cols = image.shape[:2]
        M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
        return cv2.warpAffine(image, M, (cols, rows))

    rotated_image = rotate_image(image, 45)
    show_image("Rotated Image (45 degrees)", rotated_image)

    # Reflection Function
    def reflect_image(image):
        return cv2.flip(image, 1)  # Flip horizontally

    reflected_image = reflect_image(image)
    show_image("Reflected Image (Horizontal Flip)", reflected_image)

    # Cropping Function
    def crop_image(image, x, y, width, height):
        return image[y:y+height, x:x+width]

    cropped_image = crop_image(image, 50, 50, 200, 200)
    show_image("Cropped Image", cropped_image)

    # Shearing Function (X-axis)
    def shear_image_x(image, shear_factor):
        rows, cols = image.shape[:2]
        M = np.float32([[1, shear_factor, 0], [0, 1, 0]])
        return cv2.warpAffine(image, M, (cols + int(shear_factor * rows), rows))

    sheared_image_x = shear_image_x(image, 0.3)
    show_image("Sheared Image (X-axis, factor=0.3)", sheared_image_x)
```